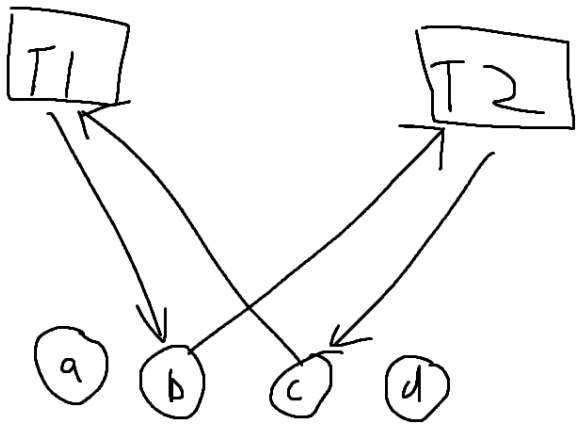
Laboratorio harjoitus

1. Ohjelma ei toimi oikein koska tuottaja tuottaa uusia arvoja liian nopeasti ja korvaa vanhemmat arvot, kun puskuri on täynnä. Ohjelman saa toimimaan halutulla tavalla, tuottajaan lisää luentokalvojen mukaisen while loopin, joka estää sitä tuottamasta silloin kun puskuri on täynnä.

2. Ohjelma estää taskeja pääsemästä käsiksi yhtä aikaa yhteiseen muuttujaan mutex-lukon avulla. Ohjelmassa critsem taskien while (!stop) -looppien alussa ennen kriittiselle alueelle siirtymistä mutex-lukko laitetaan kiinni, jos se on auki tai jäädään odottamaan mutex-lukon avautumista, jos se on kiinni. kun taski on suorittanut toimintansa kriittisellä alueella se avaa mutex-lukon, jolloin seuraava vuoroaan odottava taski pääsee suorittamaan prosessinsa kriittisellä alueella.

3. Ongelma voidaan ratkaista esimerkiksi leipomoalgoritmilla tai Petersonin algoritmilla. Petersonin algoritmi vaikuttaa helpommalta toteuttaa, joten ratkaisin ohjelman crit ongelman sillä.

4. Ohjelmassa syntyy lukkiutuma, koska taskit jäävät odottamaan resursseja toisiltaan. Kumpikaan taski ei voi pääse suoriutumaan ilman tarvittavia resursseja, joten ne jäävät odottavat, että resursseja vapautuisi. Kuva alla.



5. Lukkiutumista ei tapahdu, koska sen sijaan että taski 1 varaisi heti alussa resurssin, jonka taski 2 haluaa toisena, se varaa ainoan resurssin, jota taski 2 ei tarvitse. Tällöin taski 1 jää odottamaan resurssin b vapautumista jolloin taski 2 voi vapaasti varata resurssin c ja sitten resurssin d. Tämän jälkeen taski 2 vapauttaa resurssit, jolloin taski 1 saa varattua resurssin b ja taski 2 jää odottamaan resurssin b vapautumista. Taskit vuorottelevat tällä tavoin menemättä lukkiutumaan. Toisin sanoen ongelma on vältetty muuttamalla järjestystä, jolla taskit varaavat resursseja. Ongelman ratkaisuun

on käytetty luentokalvoista alaotsikolla ”Tunnistetaan lukkiumavaara ja vältetään sellaista resurssien allokointia, joka johtaa lukkiutuman syntymiseen” löytyvää ratkaisua.

6. Äitiprosessi luo lapsiprosessin ja tulostaa ”I created a child (Pid = xxxx)”. Tämän jälkeen äitiprosessi lähettää lapsiprosessille paketin. Lapsiprosessi ilmoittaa saapuneesta paketista ja kertoo paketin koon ja sisällön. Tämän jälkeen lapsiprosessi terminoi itsensä. Äitiprosessi huomaa, että lapsiprosessi terminoitui, jolloin äitiprosessi terminoi itsensä.

    a) Lapsiprosessi tulostaa ”Child of process[yyyy] in execution”, missä yyyy on äitiprosessin pid, joten lapsiprosessi tietää äitiprosessin pid:n. Äitiprosessi puolestaan tulostaa ”I created a child (Pid = xxxx)”, joten äitiprosessi tietää lapsiprosessin pid:n.
    b) Prosessi tarvitsee vastaanottajan IP-osoitteen ja portin numeron socket-kommunikoinnissa.
    c) Lapsiprosessissa fork palauttaa arvon 0 ja äitiprosessissa fork paluttaa lapsiprosessin pid:n. Joten kun fork paluttaa arvon 0 prosessi tietää olevansa lapsiprosessi.
    d) Rivi ”printf("Process[%d]: Mom's port is %d\n", getpid(), ntohs(their_addr.sin_port));” lisätty lapsiprosessin koodiin.